Toward optimizing disk-to-disk transfer on 100G networks

Eun-Sung Jung, Rajkumar Kettimuthu, and Venkatram Vishwanath
Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439
Email: {esjung,kettimut,venkatv}@mcs.anl.gov

***Abstract*—The recent emergence of ultra high-speed networks up to 100 Gbps has posed numerous challenges and led to many investigations on efficient protocols to saturate 100 Gbps links. Previous studies showed that RDMA over Converged Ethernet (RoCE) is efficient in terms of CPU load and achievable transfer bandwidth. However, end-to-end data transfers involve many components, not only protocols, affecting overall transfer performance. These components include a disk I/O subsystem, additional computation associated with data streams, and network adapter capacities. For example, achievable bandwidth by RoCE may not be implementable if disk I/O or CPU becomes a bottleneck in end-to-end data transfer. In this paper, we first model all the system components involved in end-to-end data transfer as a graph. We then formulate the problem whose goal is to achieve maximum data transfer throughput using parallel data flows. Our proposed formulations and solutions are evaluated through experiments on the ESnet 100G testbed. The experimental results show that our approach is several times faster than Globus Online – 8x faster for datasets with many 10MB files and 4x faster for datasets with many 100MB files.**

I. Introduction

Scientific workflows are getting more data-intensive since technology advances such as sensor data resolution make abundant data available for analysis. In addition, distributed high-performance computing resources, such as supercomputers, make data movement among geographically distributed sites a major factor that should be taken into account for efficient and reliable scientific workflow management. End-to-end data transfers involve many components affecting the overall transfer performance. Disk to disk data transfers start with disk reads, go through data transfer over network, and end up with disk writes. But the process is not simple. For example, disk reads may involve multiple disks on which data is distributed randomly or with some rules.

The recent emergence of high-speed network up to 100 Gbps poses considerable challenges and lead to many investigations on efficient protocols to saturate 100 Gbps links. Previous studies have focused on tuning and benchmarking various protocols to determine what protocols are suitable for ultra high-speed networks. Among those protocols, RoCE is shown to be very efficient in terms of CPU load and achievable transfer bandwidth.

However, reduced overhead obtained from network protocols may not directly lead to overall end-to-end data transfer performance gains. For example, for datasets with lots of small files (which is the case for many scientific datasets [1]), more time is spent exchanging control messages leading to lower end-to-end throughput [2]. Smaller block size can also seriously impact the performance. The common issue in these two problems is that they increase CPU load and can be improved by using more CPUs in parallel. Moreover, multiple network interface cards (NICs) with different capabilities, such as RoCE-capable NIC, can be exploited.

In this paper, we focus on optimizing CPU loads and parallel flows in end-to-end data transfers. More specifically, we show how the throughput for datasets with lots of small files (LOSF) can be improved through optimizing the number of parallel flows under constraints of CPU, disk I/O, and so on. For many applications, the individual file sizes in the data set are still small with respect to increasing bandwidth-delay products even though the total volume of the datasets have increased significantly in the past decade. For large files, the approach of splitting a file into multiple chunk and transferring the chunks simultaneously improve the performance. However, the same approach does not work with small files, or even hurt the performance. In this paper, we show that our approach improves the performance significantly by optimizing parallel data flows.

We first empirically evaluate the throughput and CPU load depending on disk I/O and protocols. We model system components using linear or quadratic regression model. We then develop disk I/O flow and protocol scheduling algorithms based on the developed models.

The remainder of this paper is organized as follows. In Section II, we discuss the related work. In Section III, we describe the target system and associated challenges we are facing. In Section IV, we describe the preliminary system modeling for the optimization problem formulations and we formulate the problem formally. Our proposed formulation and solutions are evaluated through extensive experiments in Section V. In Section VI, we summarize our work and briefly discuss future work.

II. Related Work

Recently many studies have been conducted on new 100G high-speed networks. In [3], various data transfer middleware such as GridFTP [4] and SRM [5] has been evaluated to determine whether they can saturate a 100G network link. The results in [3] show that they can achieve 80-90 Gbps in case of memory-to-memory data transfer, where the system's RAM buffer cache is big enough to hold the entire dataset, so the dataset is loaded into memory before data transfer.

Such performance improvements have resulted from several research areas. First, the attempts to optimize network protocols have brought enhanced network throughput. Globus eXtensible Input/Output System (XIO) [6] provides a framework on which applications do not have to care about what protocols are best for the data transfer over the networks. RDMA-based protocols have been evaluated compared with common protocols such as TCP for high-performance data transfers [7]. The results show that RDMA-based protocols can achieve 10 Gbps data transfer with much lower operating system overheads. In [8], efficient data transfer protocols based on RDMA (e.g., RDMA-based ftp) has been proposed. Another research area focuses on exploiting multiple flows to achieve high-performance data transfer. GridFTP also use pipelining [2] and concurrency [9], to offset protocol overhead for small files, and recently it has been shown recently system tools simply adapted for parallel file system I/O could achieve much better performance [10].

However, because of the lack of a holistic approach to end-to-end data transfer, achieving high-performance data transfer is difficult in varying hardware and software environments. End systems are becoming more and more complex and heterogeneous. System hierarchy is becoming deep and complex with multi-dimensional topologies. Applications must be smart enough to take advantage of parallelism in various subsystems. So far, manual hardware and software tuning have been needed in order to figure out what configurations are to be set to meet the required data transfer rate. In this paper, we address this problem by modeling system components involved in data transfer and solving mathematically formulated problems.

III. Problem Statement

In this section, we describe the target system and associated challenges we are facing.

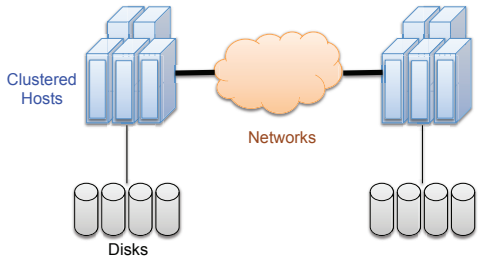

Fig. 1. Overall system configuration

The overall system configuration we target throughout this paper is depicted in Fig. 1. Two clusters of hosts are connected through networks consisting of multiple domains. Any host in one cluster can send data to any host in the other cluster over the networks. One host has multiple CPU cores and is connected to multiple disks. In addition, more than one NIC may exist for utilizing multiple streams or different network paths.

Figure 2 shows a more detailed system diagram. From the perspective of one data flow, it starts from a storage, and it arrives at a host on the other side after passing through CPUs and being transferred over network paths. The data flow can take different paths depending on which disk the data reside in, what core is allocated for data processing, and what network path is set up for transfer.

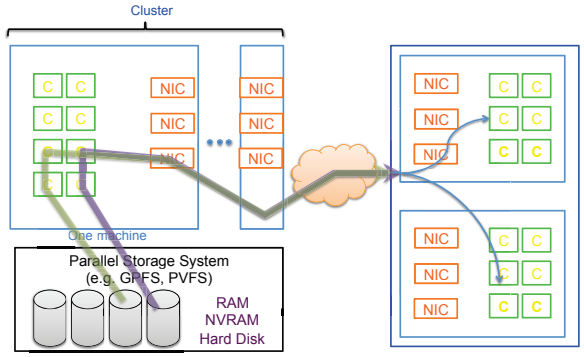

Fig. 2. Detailed system components

As described in Section II, the issues posed by ultra high-speed networks such as 100 Gbps networks are addressed in three system components by using either parallelism or optimized software stack. For disk I/O, coordinated parallel disk streams help improve the data flow throughput. If CPU computations are involved in data flows, allocating separate CPU cores per data flow may help improve the performance. For example, in case data encryption is additionally required for secure data transfer, parallel computing associated with parallel data flows will dramatically improve the throughput. Data transfer over networks also requires sophisticated control or protocol optimization.

However, addressing the throughput issues in each component separately may lead to waste of unnecessary resources and performance degradation since the bottleneck of three components dominates the overall performance.

We propose data flow optimization algorithms to maximize the throughput while taking into account any constraints inherent in the system.

IV. End-to-End Transfer Optimization

In this section, we describe how to model system components relevant to end-to-end data transfer and we formulate the problem mathematically based on models.

A. System Modeling

In this section, we discuss how we can model each component of the system so that we can develop optimization formulations to solve.

The overall system can be modeled as a graph as shown in Figure 3. In the graph, there are five classes of nodes, and

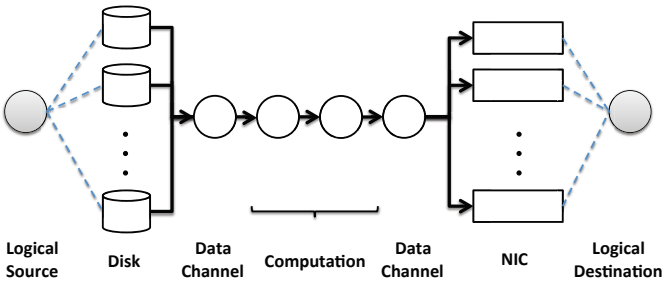

Fig. 3. Data flow graph model

edges that link adjacent nodes. The five classes of nodes are disk node, data channel, computation node, NIC , and logical node. A node is not associated with any attribute, but an edge is associated with attributes describing a node's characteristics. Data channel nodes reflect contention among data flows. For example, if all disks are connected to only one disk interface adapter, maximum disk throughput may not scale linearly as the number of disks increases due to data contention. Logical nodes are inserted for explicit data flow start and end in a graph model. The CPU cores are not expressed explicitly as a node but are put implicitly as costs on edges and constraints in the resulting formulations.

Two attributes are assigned on an edge. One is *capacity/bandwidth* of a source node. The other is *cost* of a data flow on the edge. Both attributes can be either a constant value or a function of some parameters originating from underlying system behaviors. Depending on two end nodes linked by an edge, it has different attributes. First, the edge linking from a logical start node to a disk node, *logical edge*, is a logical link with unlimited bandwidth and zero cost function. Second, the edge linking from a disk or data channel node to any other node, *disk edge*, represents a disk I/O path from a disk or data channel. Third, the edge linking from a computation node to an another computation node or a NIC node, *compute edge*, represents a data flow going through computations such as GridFTP and compression computation. Fourth, the edge linking from a NIC node to a logical end node, *network edge*, represents a network path from a source node to a destination node. Each edge is associated with a bandwidth function and a cost function. A bandwidth function and a cost function of an edge describe the performance throughput and CPU resource consumption of a source node, respectively. In the following subsections, we describe each edge's attributes and associated modeling in details.

1) Disk modeling: Disk edge

Disk edge is associated with disk capacity/bandwidth and CPU load related to disk I/O operations. Even though many parameters such as disk cache size are involved in disk I/O bandwidth, the number of data flows per disk is the most important variable assuming that other parameters are fixed and not adjustable.

Equation (1) computes utilization of a disk as a function of number of processes and disk access probability [11]. Here p is a ratio of request data size and the stripe size of a RAID disk. If we assume that file size or request data is bigger than the stripe size of a disk, p can be substituted by 1. The resulting equation is $U \simeq \frac{1}{1+\frac{\gamma}{L}}$, which means the disk utilization increases to some extent as the number of processes increases. γ is a constant to take into account other factors in disk performance such as bloack size and disk cache.

$$\begin{aligned} U &\simeq \frac{1}{1+\frac{1}{L}(\frac{1}{p}-1+\gamma)} \\ U &: \text{ Utilization} \\ L &: \text{ Number of processes issuing requests} \\ p &: \text{ Probability that a request will access a given disk} \\ \gamma &: \text{ Empirically calibrated value} \end{aligned} \tag{1}$$

The disk throughput can be determined by Equation (2) in which the disk utilization in Equation 1 is multiplied by $\frac{N \cdot SU}{E(S)}$. The equation can be rearranged as Equation (3) after substituting $\frac{N \cdot SU}{E(S)}$ by $\frac{\alpha}{L}$, where $\alpha = N \cdot SU$, since $E(S)$, the expected service time of a given disk request, is proportional to L. We can determine α and γ in Equation (3) through experimental values. The cost function for the edge is assumed to be zero since CPUs sit idle until a read/write operation is done and any file operation overhead can be attributed to computation nodes.

$$\begin{aligned} T &= \frac{U \cdot N \cdot SU}{E(S)} \\ T &: \text{ Throughput} \\ U &: \text{ Utilization} \\ N &: \text{ Number of disks in a RAID disk} \\ SU &: \text{ Stripe size} \\ S &: \text{ Service time of a given disk request} \end{aligned} \tag{2}$$

$$\begin{aligned} T &= \frac{1}{1+\frac{\gamma}{L}} \cdot \frac{\alpha}{L} = \frac{\alpha}{L+\gamma} \\ \alpha, \gamma &: \text{ Empirically calibrated value} \end{aligned} \tag{3}$$

If the source node is data channel node, the disk edge can be associated with this bandwidth function when the data channel node has fan-in disk nodes, or can be associated with infinity bandwidth when the data channel node has fan-out nodes.

Equation (3) will be used as bandwidth function B^n_{lk} in Section IV-B and approximated by a linear/quadratic function for linear programming solver such as cplex [12].

2) Computation modeling: Compute edge

The edge whose source node is a computation node has attributes of linear bandwidth and cost functions. The bandwidth function is a function of the number of flows as in Equation (4), and the cost function can be defined as in Equation (5).

$$\begin{aligned} T &= \alpha n_s + \beta \\ n_s &: \text{ Number of parallel data transfer streams} \\ \alpha, \beta &: \text{ Empirically calibrated value} \end{aligned} \tag{4}$$

$$\begin{aligned} C(r) &= \alpha r \\ C(\cdot) &: \text{ CPU load} \\ r &: \text{ Data flow rate} \\ \alpha &: \text{ Empirically calibrated value} \end{aligned} \tag{5}$$

Equation (4) and (5) will be used as bandwidth function $B^c_{lk}()$ and cost function $C^c_{lk}()$, respectively, in Section IV-B.

3) Network modeling: Network edge

The edge linking a NIC node and a logical destination node has attributes of a throughput function and a cost function. In order to simplify the problem, only TCP is considered and a NIC is assumed to have a preassigned protocol property associated with corresponding throughput function.

Several throughput models for parallel TCP streams have been proposed to predict the performance. The simplest model is proposed in [13] and given by Equation (6).

$$\begin{array}{lll} T & \leq & \min\{NC, \frac{MSS \times c}{RTT} \cdot \frac{n_t}{\sqrt{p}}\} \\ & T: & \text{Achievable throughput} \\ & NC: & \text{Capacity of NIC} \\ & MSS: & \text{Maximum segment size} \\ & RTT: & \text{Round trip time} \\ & p: & \text{Packet loss rate} \\ & n_t: & \text{Number of parallel data transfer streams} \end{array} \quad (6)$$

Since $\frac{MSS \times c}{RTT} \cdot \frac{1}{\sqrt{p}}$ is a constant, Equation (6) can be rearranged as $\alpha \cdot n_t$ where $\alpha = \frac{MSS \times c}{RTT} \cdot \frac{1}{\sqrt{p}}$.

The cost function for TCP is given by Equation (7).

$$\begin{array}{lll} C(r) & = & \alpha r \\ & C: & \text{CPU load} \\ & r: & \text{Data flow rate} \\ & \alpha: & \text{Empirically calibrated value} \end{array} \quad (7)$$

Equation (6) will be used as bandwidth function B^n_{lk}, and Equation (7) will be used as cost function C^p_{lk} in Section IV-B.

B. Problem Formulation

In order to simplify the problem, the following assumptions are made:

- There is only one machine at each end. (Cluster-level modeling and formulation will be future work)
- There is a dedicated network path between the sender and the receiver machine.
- The data rates of all parallel data flows are same. This means that the total data rate (and data I/O load) is evenly distributed over current parallel flows. Even though disks attached to the machine may have slightly different capacities, we assume homogeneous disk resources in this paper.
- A sender and a receiver have similar hardware such that optimization on the sender side is sufficient for end-to-end data transfer optimization.
- Number of parallel transport protocol (TCP/RoCE) flows can be greater thanthe number of parallel data flows from disks.

The last assumption means that the system can automatically adjust the number of network transfer streams if one network transfer stream is not enough, in order to accommodate output data rate from computation nodes. For example, GridFTP [4] use multiple logical TCP flows, called parallelism, per one data stream to overcome the limitation of TCP protocol in high-bandwidth high-latency networks.

The overall problem-solving procedure is as follows.

- Compute parameters of capacity functions based on empirical data.
- Formulate the modified multicommodity flow problem based on the capacity/cost functions on edges.
- Find a solution including the number of parallel flows, the number of required CPUs, and the number of NICs using linear programming solver, cplex [12].
- Determine the number of parallel TCP/RoCE flows based on the amount of flows on network edges.

The graph model as in Figure 4 can be formally represented by a graph $G = (V, E)$, where V is a set of vertices, and E is a set of edges.

Table I gives a list of notations for mathematical formulations, and the complete formulation is described in Fig. 5. The formulation in Fig. 5 is mixed-integer convex programming (MICP) since n_s, the number of data streams, is integer and bandwidth function B^d_{lk} is approximated by quadratic functions.

The objective function is given in Expression (8), which is to maximize overall throughput of data transfer. Expression (10) set the range of the number of data streams per disk. Expression (11) is a bandwidth/capacity constraint on the edges, where r_{lk} denotes data rate on an edge (l, k) and bandwidth functions is chosen depending on the edge type. The flow conservation constraint given by Equation (12) ensures that the sum of incoming data rates should be same as that of outgoing data rates at every node. In special cases such as compression computation, the sum of outgoing data rates can be a fraction of that of incoming data rates. Expression (13) and (14) constrain the total outgoing data rates from the logical source and to the logical destination node to be greater than or equal to T, which is to be maximized. In this way, we can get the solution that maximizes the overall data throughput. The computation constraints by the number of CPU cores in the system and the number of data flows is given by Expression (15). Note that the formulation assume a circumstance where the number of data flows per disk is same, but the formulation can be easily extended to reflect different number of data flows per disk by assigning separate variables per disk.

V. Experimental Evaluation

We have conducted experiments on an ESnet 100G testbed in two locations: NERSC (Oakland, CA) and Argonne (near Chicago, IL). Fig. 6 shows the detailed configuration of the testbed. At NERSC, there are 5 hosts of three different hardware configurations. We have used either nersc-diskpt-1 or nersc-diskpt-2, which have Intel Xeon Nehalem E5650 (2 x 6 = 12 cores), multiple 10G NICs, and 4 RAID 0 sets of 4 drives. On the other hand, there are 3 hosts without disk arrays at Argonne. Likewise, we have used one host out of 3 hosts at Argonne. These hosts have 2 AMD 6140 (2 x 8 = 16 cores) and multiple 10G NICs, but do not have RAID disks. However, the host at Argonne has only local disk, which is slow (i.e. 300MB/s) and thus can not saturate a 10G link. For such reasons, we conducted disk to memory tests where all data flows arriving at the host at Argonne will be directed to

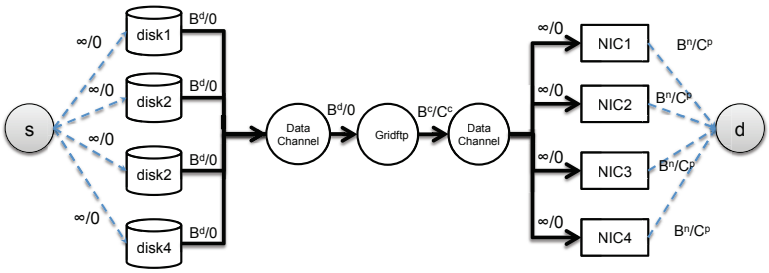

Fig. 4. NERSC host graph model

TABLE I. NOTATION FOR PROBLEM FORMULATION

Notation	Description
V_s	Logical source node
V_d	Logical destination node
N_d	Number of disks
N_c	Number of CPU cores
n_s	Number of data streams per each disk; integer variable
n_{lk}^t	Number of parallel TCP streams on an edge (l,k)
r_{lk}	Data rate on an edge (l,k)
$B_{lk}^d(n_s)$	Disk capacity/bandwidth of V_l, a disk node, associated with an edge (l,k)
$B_{lk}^c(n_s)$	Computation capacity of V_l, a computation node, associated with an edge (l,k)
$B_{lk}^n(n_t)$	Maximum network capacity/bandwidth of V_l, a NIC node, associated with an edge (l,k)
$C_{lk}^c(r_{lk})$	CPU/Computation cost of V_l, a computation node, associated with an edge (l,k)
$C_{lk}^p(r_{lk})$	CPU/Computation cost related to network protocol on V_l, a NIC node, associated with an edge (l,k)

Objective

$$\text{maximize } T \tag{8}$$

Subject to:

$$r_{lk} \geq 0,\ (l,k) \in E \tag{9}$$

$$0 \leq n_s \leq M_s, M_s \text{ is maximum number of data streams.} \tag{10}$$

$$r_{lk} \leq \begin{cases} B_{lk}^d(n_s),\ (l,k) \in E, & \text{if } V_l \text{ is a disk node} \\ B_{lk}^c(n_s),\ (l,k) \in E, & \text{if } V_l \text{ is a computation node} \\ B_{lk}^n(n_t),\ (l,k) \in E, & \text{if } V_l \text{ is a NIC node} \end{cases} \tag{11}$$

$$\sum_{k:(l,k)\in E} r_{lk} - c \sum_{k:(k,l)\in E} r_{kl} = 0,$$

$$l \neq s_j, l \neq d_j, c = \begin{cases} \text{compression ratio}, & \text{if } l \text{ is compression node} \\ 1, & \text{otherwise} \end{cases} \tag{12}$$

$$\sum_{k:(s,k)\in E} r_{sk} - \sum_{k:(k,s)\in E} r_{ks} \geq T \tag{13}$$

$$\sum_{k:(k,d)\in E} r_{ks} - \sum_{k:(d,k)\in E} r_{dk} \geq T \tag{14}$$

$$\sum_{k:(l,k)\in E} C(r_{lk}) \leq \min\left(N_c \times 100, n_s \times N_d \times 100\right) \tag{15}$$

Fig. 5. Multiple flow determination algorithm

/dev/nulll or nullfs, and we assume that the host at Argonne has same hardware performance as that of the NERSC host.

We have chosen lots of small files (LOSF) dataset as an exemplary dataset for our and focus on optimizing the end-to-end data transfer rates for LOSF datasets. We use three

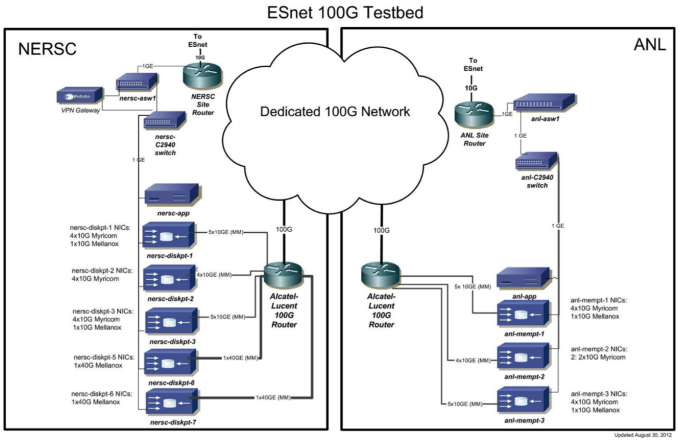

Fig. 6. ESnet 100G testbed

different datasets – one with ten thousand of 1 MB files, one with one thousand 10 MB files and one with one hundred 100 MB files, such that total amount of data would be 10 GB. The files were synthetically generated using */dev/urandom* in Linux.

To measure the disk performance, we use *dd* and *iozone* as disk I/O benchmark tools. In addition, we use *nmon* and *netperf* as benchmark tools to measure CPU load and network performance, respectively.

A. Subsystem Tests for Model Parameter Setting

We first conducted basic disk I/O performance tests using *dd* disk utility to obtain baseline performance of disk throughputs. Fig. 7 shows the disk read throughputs of 4 RAID sets attached to hosts at NERSC, and. The theoretical upper limits of each RAID disk is around 1.2 GB/s since the RAID disk is composed of four disks with 300 MB/s read performance.

Even though there are performance variances among disks, we ignore the variances for simplicity in this paper.

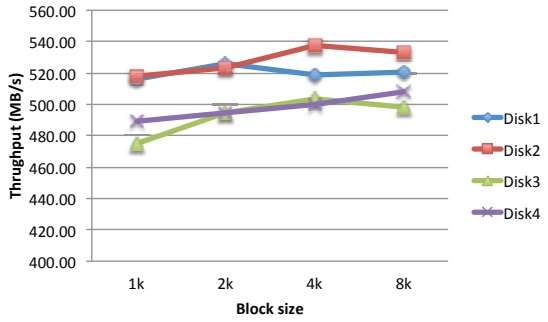

Fig. 7. Disk throughput at NERSC using *dd*: similar disk throughput with varying block size

Next, we measured the multithread disk read performance depending on file size and the number of threads to determine the value of α, γ in Equation (3). Fig. 8 shows that disk throughput decreases as the number of streams increases and is irrelevant to the file size. We conducted experiments in case of sequential disk read. Equation (3) determined by these results would be $B^d_{lk}(\cdot)$ in Fig. 5 where L equals n_s and l is a disk node.

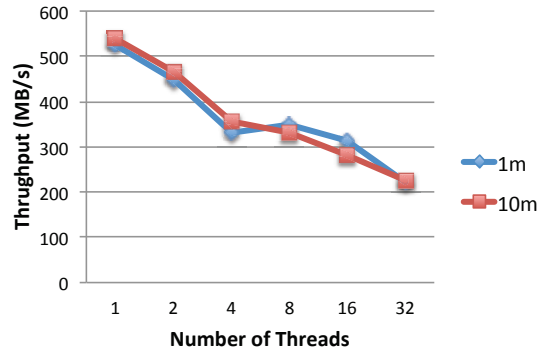

Fig. 8. Disk throughput at NERSC using *iozone*: decreasing disk throughput with increasing number of data streams

However, as Fig. 9 shows, the aggregate disk throughput using multiple disks does not scale linearly due to channel contention. We model the channel contention using a data channel node and associated bandwidth function, as in Equation, 3.

Next, we measured application throughput while varying the number of data streams. Even though the data movement tool GridFTP is the only application used for the end-to-end data transfers, we measured *tar* application throughput as well as GridFTP to help better understand characteristics of application throughput. The application throughput characteristics of GridFTP is described in Section V-B. Fig. 10 shows how *tar* application throughput is affected by the number of data flows. Apparently, as seen in Fig. 10, in the case of 1 MB files, its throughput is increasing as a function of n_s, the number of data flows. This behavior can be modeled by Equation (4) based on linear regression model or nonlinear functions such as quadratic functions. Even though we chose to use linear regression for simplicity in this paper, the nonlinear function can also be used as long as mixed-integer convex programming

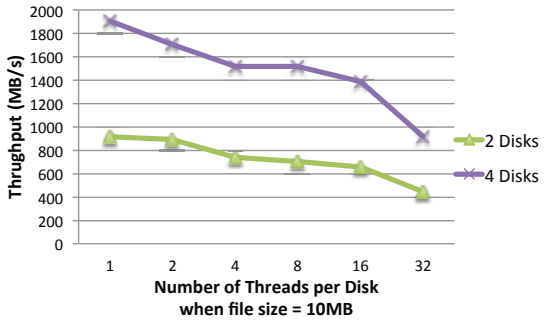

Fig. 9. Multiple disk throughput at NERSC using *iozone*: increased throughput using multiple disks, but slightly under the total sum of individual disks

(MICP) solvers can find a solution within a reasonable time. The behavior for 10 MB files in *tar* application can not model by Equation (4), but it is only because the disk throughput is limited by around 500MB/s, and *tar* throughput hit the limit when the number of data flows is greater than 1. In this case, we need to replace the disk by a high-end disk or memory in order to measure the application throughput accurately, and will get linear model as for 10 MB, too.

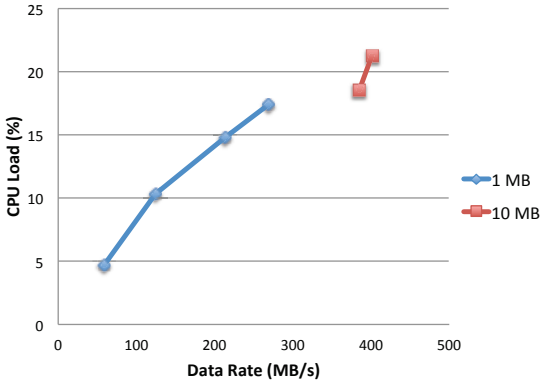

Fig. 10. Application throughput (tar): increasing throughput with increasing number of data streams until disk throughput becomes a bottleneck

Fig. 11 shows how *tar* application CPU load is affected by the processed data rate. The experimental results of 1 MB files and 10 MB files are plotted with the x-axis as data rate and the y-axis as CPU load. In the case of 10 MB, redundant data have been removed for plotting. Regardless of file size, we can see that CPU load is increasing as a function of r, data rate, and can also be approximately by Equation (5) based on linear regression model. In a similar way, we can model any applications including GridFTP and compression.

Regarding network edges, Fig. 12 and 13 show the throughput and CPU load of TCP protocol, respectively. These TCP performance results are measured by *netperf*, and memory-to-memory transfer over a 10G NIC. Fig. 12 shows that network transfer throughput is saturated with 3 TCP streams and is near the full capacity of the 10G NIC. Obviously, these protocol behaviors can be modeled by Equation (6) and (7).

B. End-to-End Tests

The end-to-end data transfer results using GridFTP are shown in Fig. 14 and 15. To eliminate the disk bottleneck

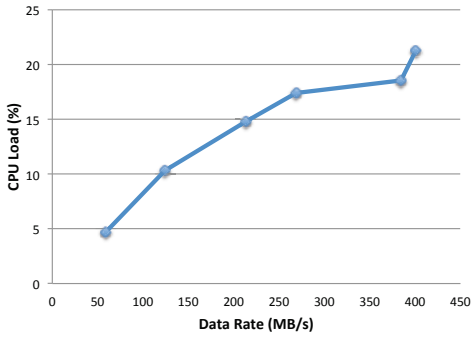

Fig. 11. CPU load (tar): linearly increasing CPU load with increasing data rate

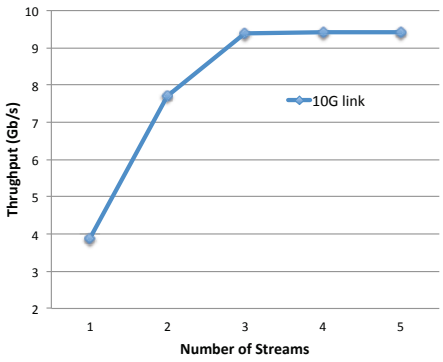

Fig. 12. TCP throughput: three TCP streams can saturate 10G network link in the case of memory-to-memory data transfer

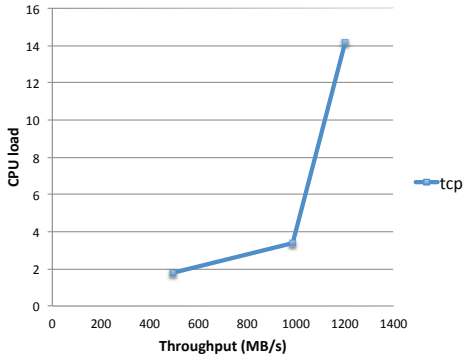

Fig. 13. TCP CPU load: increasing CPU load with increasing data rate

constraints on hosts at Argonne, we performed disk-to-memory data transfer experiments. The symbolic link files linked to */dev/null*, which correspond to names of files to be transferred, are generated in advance. GridFTP provides 4 types of options for data transfer optimization: (1) *-cc*: concurrent FTP connections, (2) *-p*: number of parallel data connections, (3) *-pipeline*: optimization to hide control message exchange latency in case of multi-file ftp transfers, and (4) *-fast*: fast directory listing and data channel reuse. In this experiment, we use only the *-fast* option since the other options are related to the number of data streams to be determined by our optimization formulations. *-cc* specifies the number of threads to read files from disks when transferring multiple files. *-p* and *-pipeline* do not relate to the number of threads, but to the number of connections and the overhead in control message exchanges. Fig. 14 shows that the throughputs of 1 MB/10 MB files using single disk increase up to 8 data flows while 100 MB files is saturated at 2 data flows and even declines as the number of data flows grows. That is obviously due to disk throughput limit and disk throughput degradation with the bigger number of data flows, as shown in Fig. 8.

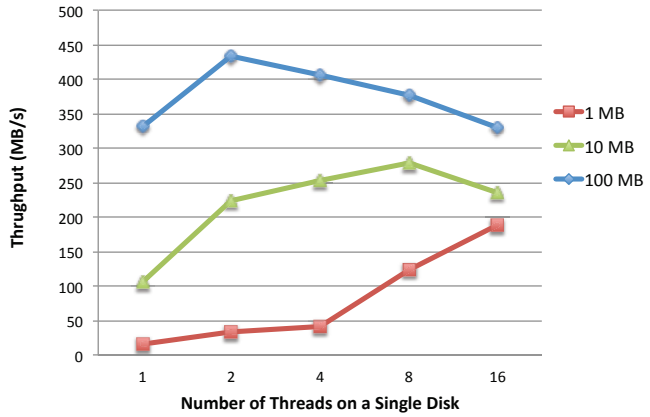

Fig. 14. GridFTP throughput: increasing throughput until disk throughput or data contention among multiple data streams becomes a bottle

In the case of multiple disks, its throughput could increase up to 1.2 GB at 4 data flows in case of 100 MB files as in Fig. 15. The maximum throughput is limited by a capacity of 10 Gbps NIC and may be improved by multiple NICs.

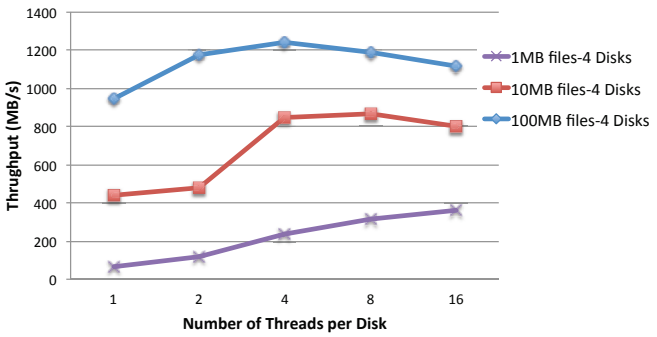

Fig. 15. Multiple disk GridFTP throughput: increased throughput using multiple disks

C. Results and Discussion

We have compared our model-based optimization approach with three cases: (1) GridFTP with only *-fast option*, (2) GridFTP with auto-tuning optimizations currently used by Globus Online [14], and (3) Best GridFTP results obtained using manual tuning. Globus Online's auto-tuning algortihm uses different GridFTP optimization options depending on file size. If the number of files is more than 100 and an average file size smaller than 50 MB, it uses GridFTP with concurrency=2 files, parallelism=2 sockets per file, and pipelining=20 requests outstanding at once. If file size is larger than 250 MB, Globus Online uses options of concurrency=2, parallelism=8, and

pipelining=5. In all other cases, the default setting is used: concurrency=2, parallelism=4, and pipelining=10. Since our file size ranges from 1 MB to 100 MB, we use the options for <50 MB and the default case.

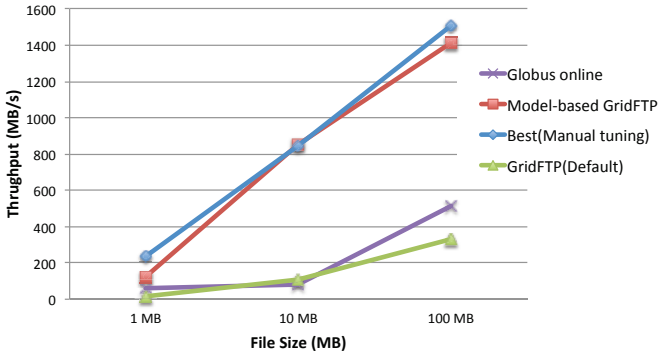

Fig. 16. Throughput Comparison

Globus Online outperforms the naive GridFTP especially in case of 1 MB and 100 MB. The performance obtained using the optimization values predicted by our model is as good as or close to the best results obtained with manual tuning. Of course, manual tuning involves trial and error and is time consuming and tedious. The best result with manual tuning and our model-based results are roughly 8 times and 4 times faster than Globus Online in case of 10 MB files and 100 MB files, respectively. It is mainly because our model can effectively identify the number of data flows based on disk throughput performance models. With *-cc=2* options, Globus Online can only utilize only two data streams from disks, which has a lot room for improvement, and cannot utilize the advantages of multiple disks. Based on models, our formulation in Fig. 5 could find the proper number of data flows, 8, 6, and 3 in the case of 1 MB, 10 MB, and 100 MB files, respectively, while the best results come from 16, 8, and 4. In addition, our formulation could find a solution suggesting using 2 NICs in case of 100m files. The experimental results in Fig. 15 shows that the overall throughput is stalled when the number of data flows per disks are 4. But this is due to a bottleneck in 10G network links, and the formulation could successfully find the solution using multiple NICs to achieve maximum throughput.

The advantages of using model-based optimization formulations are as follows: (1) it can suggest the future hardware plan optimized for overall data transfer throughput just by simulating different configurations of hardware as well as software, (2) it can be used by systems such as Globus Online and other intelligent data transfer managers to adaptively optimize transfers for varying CPU resource availability and network status, and (3) it can provide basic models for simulating bulk data movement in next generationn networks.

VI. Conclusions

We first model all the system components involved in end-to-end data transfer as a graph. We then formulate the problem whose goal is to achieve maximum data transfer throughput using parallel data flows. Our proposed formulations and solutions are evaluated through experiments on the ESnet testbed. The experimental results show that our approach is several times faster than Globus Online 8x faster for datasets with many 10MB files and 4x faster for datasets with many 100MB files. Our models and formulations are extensible to more complex cases such as more deep software stacks and more complex system architectures (e.g., cluster). Accordingly, we will continue our research toward cluster-wide session control and weighted CPU scheduler based on parameters determined by optimization formulation.

Acknowledgment

This work was supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research, under Contract DE-AC02-06CH11357.

References

[1] W. E. Johnston, "Terabit networks for extreme-scale science," U.S. Department of Energy, Tech. Rep., 2011.

[2] J. Bresnahan, M. Link, R. Kettimuthu, D. Fraser, and I. Foster, "Gridftp pipelining," 2007.

[3] A. Rajendran, P. Mhashilkar, H. Kim, D. Dykstra, and I. Raicu, "Optimizing large data transfers over 100Gbps wide area networks," Nov. 2012.

[4] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster, "The globus striped GridFTP framework and server," in *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, ser. SC '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 54–64.

[5] A. Shoshani, A. Sim, and J. Gu, "Grid resource management," J. Nabrzyski, J. M. Schopf, and J. Weglarz, Eds. Norwell, MA, USA: Kluwer Academic Publishers, 2004, pp. 321–340.

[6] W. Allcock, J. Bresnahan, K. Kettimuthu, and J. Link, "The globus extensible input/output system (XIO): a protocol independent IO system for the grid," in *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, Apr. 2005, pp. 8–15.

[7] E. Kissel and M. Swany, "Evaluating high performance data transfer with RDMA-based protocols in wide-area networks," in *Proceedings of the 2012 IEEE 14th International Conference on High Performance Computing and Communication*, ser. HPCC '12, Washington, DC, USA, 2012, pp. 802–811.

[8] Y. Ren, T. Li, D. Yu, S. Jin, T. Robertazzi, B. L. Tierney, and E. Pouyoul, "Protocols for wide-area data-intensive applications: design and performance issues," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '12. Los Alamitos, CA, USA: IEEE Computer Society Press, 2012, pp. 34:1–34:11.

[9] R. K. et. al, "Lessons learned from moving earth system grid data sets over a 20 gbps wide-area network," in *19th ACM International Symposium on High Performance Distributed Computing*, ser. HPDC '10. New York, NY, USA: ACM, 2010, pp. 316–319.

[10] Kolano, "Transparent optimization of parallel file system I/O via standard system tool enhancement," in *2nd International Workshop on High Performance Data Intensive Computing*, May 2013.

[11] E. K. Lee and R. H. Katz, "An analytic performance model of disk arrays," *SIGMETRICS Perform. Eval. Rev.*, vol. 21, no. 1, pp. 98–109, Jun. 1993.

[12] Http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/.

[13] T. Hacker, B. Athey, and B. Noble, "The end-to-end performance effects of parallel TCP sockets on a lossy wide-area network," in *Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM*, 2002, pp. 10–19.

[14] B. Allen, J. Bresnahan, L. Childers, I. Foster, G. Kandaswamy, R. Kettimuthu, J. Kordas, M. Link, S. Martin, K. Pickett, and S. Tuecke, "Software as a service for data scientists," *Commun. ACM*, vol. 55, no. 2, pp. 81–88, Feb. 2012.